ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

Σχολή Ηλεκτρολόγων Μηχανικών
& Μηχανικών Η/Υ

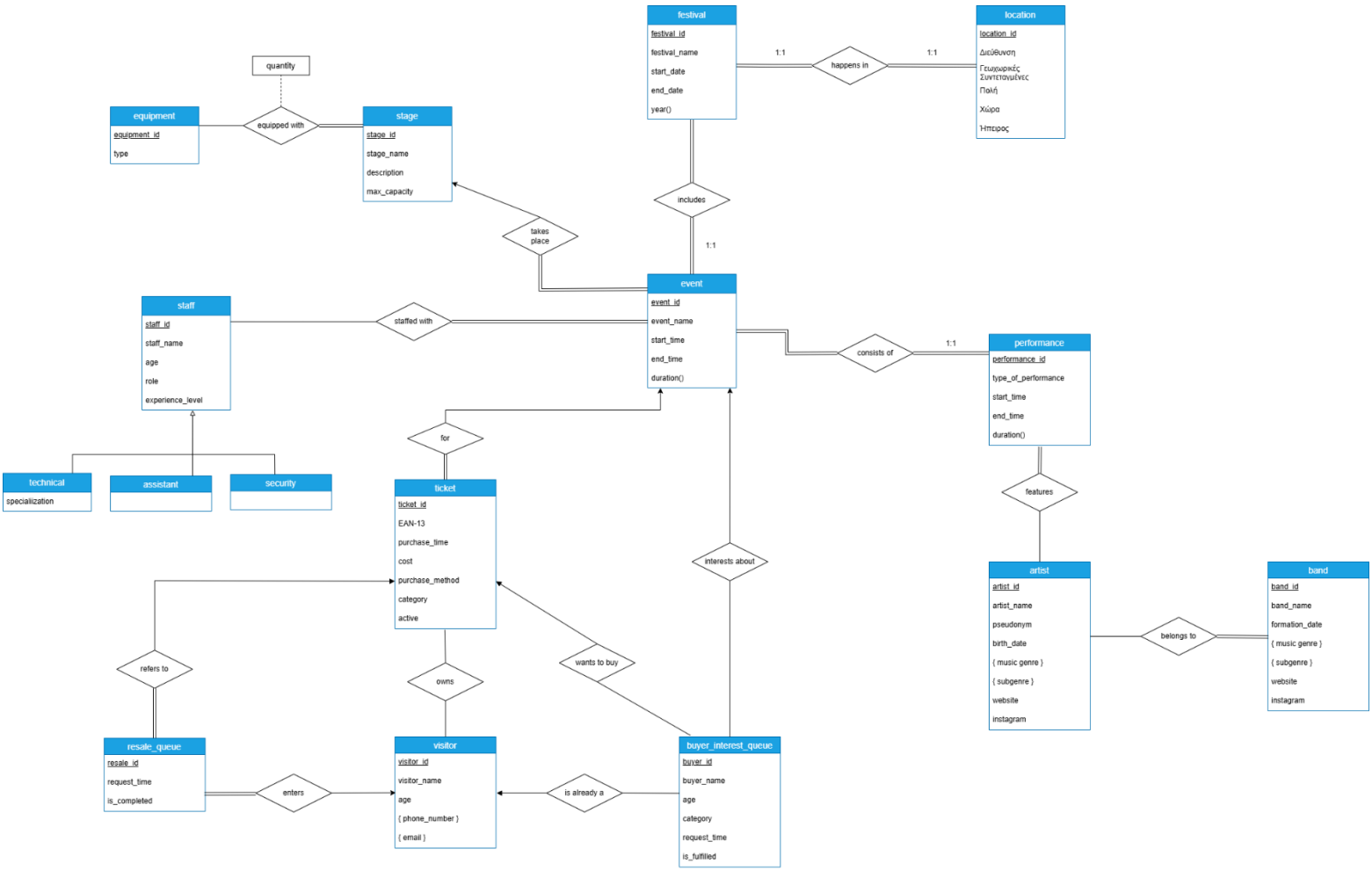
Εξάμηνο 6ο: Βάσεις Δεδομένων

Εξαμηνιαία Εργασία

ΑΝΑΦΟΡΑ

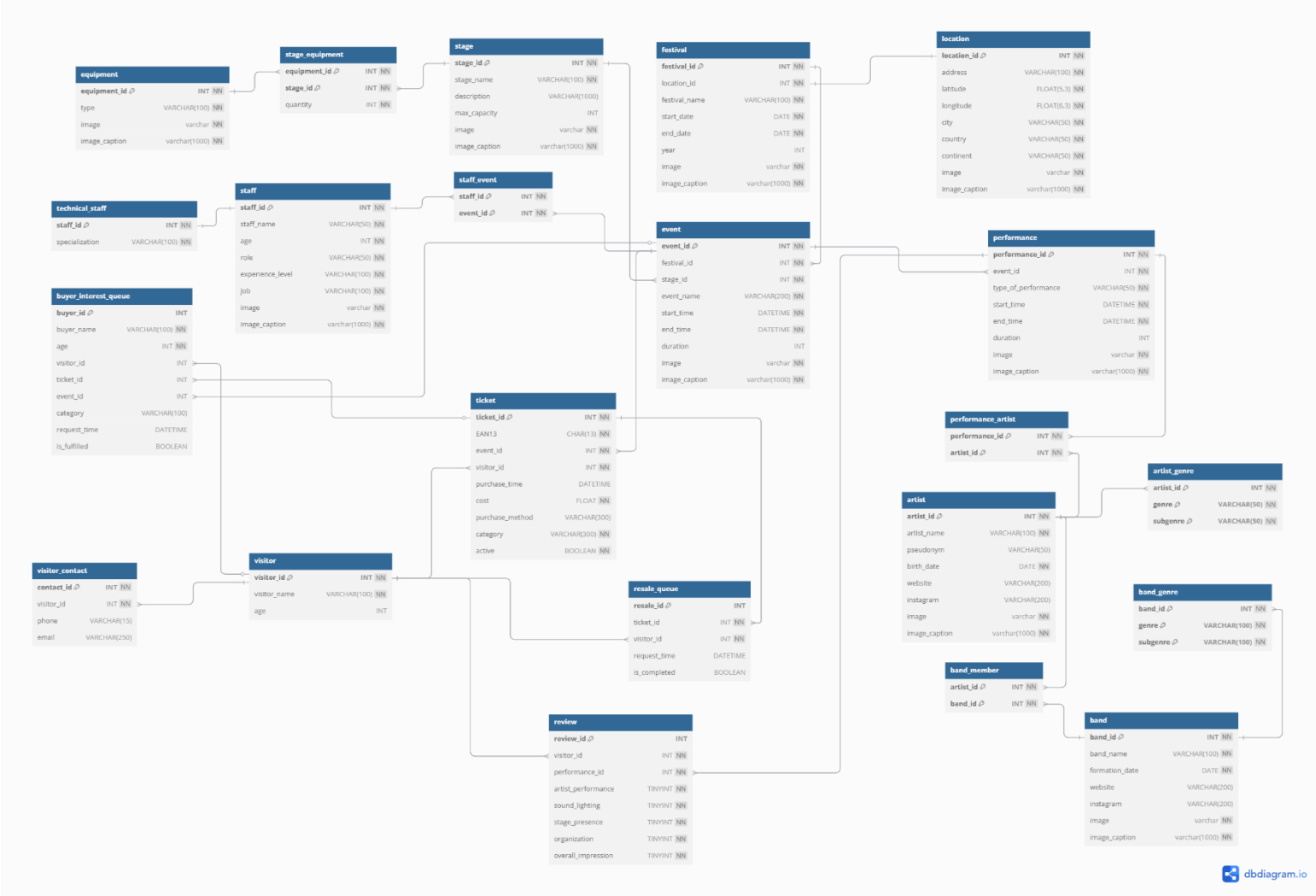
Όνομα	Επώνυμο	Α.Μ.
Μιχαήλ	Παναγόπουλος	03122061
Αθανάσιος	Φειδάκης	03122676

Διάγραμμα E/R



Σχήμα 1 - E/R Διάγραμμα

Σχεσιακό Διάγραμμα



Σχήμα 2 – Relational Διάγραμμα

Σχολιασμός Σχεσιακού Διαγράμματος

Μετά την υλοποίηση του ER διαγράμματος, όπου αποτυπώσαμε τη λογική της βάσης μας, προχωρήσαμε σε μία πιο λεπτομερή αποτύπωση των χαρακτηριστικών (attributes) των πινάκων (entities) και των σχέσεων μεταξύ τους. Τα relationships sets «αποχωρούν» και οι συσχετίσεις των πινάκων επιτυγχάνονται με τη χρήση foreign keys. Οι σχέσεις με cardinality one-to-many πραγματοποιούνται με την προσθήκη του primary key του πίνακα many ως foreign key στον πίνακα one, ενώ στην σχέση με cardinality one-to-one η επιλογή για την τοποθέτηση του foreign key είναι ελεύθερη. Αντίθετα, οι σχέσεις many-to-many απαιτούν έναν ενδιάμεσο πίνακα, ο οποίος θα έχει ως primary key το ζεύγος των primary keys (ταυτόχρονα foreign keys προς τους αρχικούς πίνακες) των πινάκων που συσχετίζει. Ο ενδιάμεσος πίνακας μας παρέχει τη δυνατότητα να προσθέσουμε επιπλέον πληροφορία για τη σχέση σε μορφή attribute. Τέλος, τα multivalued attributes του E/R διαγράμματος υλοποιούνται με τη βοήθεια επιπλέον πινάκων.

Σε κάθε κεντρικό πίνακα δημιουργείται ένα attribute id, το οποίο καθίσταται primary key και χρησιμεύει ως μοναδικό αναγνωριστικό. Το id είναι μία μεταβλητή τύπου INT UNSIGNED, αρχικοποιείται σειριακά και αυτόματα προς μεγαλύτερους ακεραίους. Όσα attributes κρίνονται απαραίτητα για τον πλήρη προσδιορισμό μίας εγγραφής τίθενται NOT NULL. Επιπλέον, στην υλοποίησή μας, τα ονόματα ανθρώπων (name) περικλείουν τόσο το first_name όσο και το last_name. Για όσα attributes αποτυπώνουν πληροφορία χρόνου χρησιμοποιήθηκαν οι τύποι δεδομένων DATE, DATETIME, YEAR κ.α.

Για το σκοπό μίας σύντομης περιγραφής του σχήματος μπορούμε να κατηγοριοποιήσουμε τις κεντρικές οντότητες ως εξής:

Διοργάνωση Φεστιβάλ: festival, event, location

- Κάθε φεστιβάλ έχει μοναδική τοποθεσία (location_id), χρονική διάρκεια (start_date, end_date) και εξάγει το year αυτόματα για χρήση σε ελέγχους.
- Το event ανήκει σε ένα συγκεκριμένο festival και γίνεται σε ένα συγκεκριμένο stage.
- Το location περιλαμβάνει γεωγραφικά στοιχεία και μοναδικό συνδυασμό συντεταγμένων.

Ανθρώπινο Δυναμικό: staff, technical_staff, staff_event

- Η γενική πληροφορία προσωπικού είναι στον πίνακα staff με διαχωρισμό ρόλων και εμπειρίας.
- Οι τεχνικοί εξειδικεύονται περαιτέρω στον πίνακα technical_staff.
- Οι αναθέσεις προσωπικού σε events καταγράφονται στον staff_event.

Σκηνή και Εξοπλισμός: stage, stage_equipment

- Αναπαριστά σκηνές με περιγραφή και μέγιστη χωρητικότητα.
- Καταγράφει εξοπλισμό ανά σκηνή και ποσότητα.

Καλλιτεχνικές εμφανίσεις: performance, artist

- Κάθε performance συνδέεται με ένα event και περιλαμβάνει πληροφορίες ώρας, διάρκειας και τύπου (π.χ. warm up, headline).
- Ο artist μπορεί να συμμετέχει σε πολλές performance μέσω του πίνακα performance_artist. Ένα performance μπορεί να συγκροτείται από πολλούς artists.

Συγκροτήματα: band, band_member

- Το band περιλαμβάνει πληροφορίες για το συγκρότημα και τις ημερομηνίες σύστασης.
- Τα μέλη της μπάντας σχετίζονται μέσω του πίνακα band_member, με αναφορά σε artist.

Είδη Μουσικής: artist_genre, band_genre

- Αντιστοιχίζονται μουσικά είδη (περισσότερα από ένα ενδεχομένως) σε κάθε καλλιτέχνη ή συγκρότημα, με κύριο και υποείδος.

Εισιτήρια και Επισκέπτες: visitor, visitor_contact, ticket, resale_queue, buyer_interest_queue

- Το ticket περιέχει πληροφορίες αγοράς, τρόπο πληρωμής και κατηγορία εισιτηρίου.
- Υπάρχει μηχανισμός για μεταπώληση εισιτηρίων μέσω resale_queue και αντιστοίχιση ενδιαφέροντος μέσω buyer_interest_queue.
- Ο πίνακας visitor_contact επεκτείνει τις πληροφορίες επικοινωνίας του επισκέπτη.

Κριτικές: review

- Οι επισκέπτες μπορούν να αφήσουν αναλυτικές αξιολογήσεις για κάθε performance με βαθμολογικά κριτήρια (καλλιτέχνης, ήχος, παρουσία, διοργάνωση κ.ά.) βάσει της κλίμακας Likert.

Επεξήγηση σχεδιαστική επιλογής

Η εκφώνηση υποδηλώνει τη δημιουργία των πινάκων πωλητής, αγοραστής, ουρά μεταπώλησης. Ωστόσο, πωλητής θα είναι πάντα κάποιος επισκέπτης επομένως μπορεί να προσδιοριστεί πλήρως μέσω του πίνακα visitor. Ακόμα, η ουρά μεταπώλησης στην υλοποίησή μας εμπεριέχει τα εισιτήρια προς πώληση, τα συσχετίζει με τον επισκέπτη-πωλητή που τα διαθέτει και φανερώνει την ολοκλήρωση της πώλησης στο πεδίο is_completed. Τέλος, ο αγοραστής, επισκέπτης ή μη, δηλώνει το ενδιαφέρον του στο buyer_interest_queue για συγκεκριμένο εισιτήριο (ticket_id) ή για κάποιο event και συγκεκριμένη κατηγορία εισιτηρίου (event_id, category) και η ικανοποίηση του αιτήματός του παρακολουθείται μέσω του πεδίου is_fulfilled.

Το σύστημα αυτό επιτρέπει την ευέλικτη διαχείριση αγορών, πωλήσεων και επαναπωλήσεων εισιτηρίων, με έναν ολοκληρωμένο τρόπο παρακολούθησης του κάθε σταδίου της διαδικασίας, διασφαλίζοντας την αποφυγή διπλών αγορών ή αναληθών συναλλαγών μέσω των περιορισμών των ξένων κλειδιών και των μοναδικών περιορισμών στα δεδομένα (π.χ. ένα εισιτήριο ανά επισκέπτη ανά εκδήλωση).

Constraints Βάσης Δεδομένων

Ο πίνακας *location* δηλώνει τα εξής constraints:

- **UNIQUE (latitude, longitude)**: Εξασφαλίζει ότι ο συνδυασμός των τιμών γεωγραφικού πλάτους και μήκους (latitude, longitude) είναι μοναδικός στον πίνακα, αποτρέποντας διπλές καταχωρήσεις για την ίδια τοποθεσία.
- **CHECK (latitude BETWEEN -90.000 AND 90.000)**: Ελέγχει ότι η τιμή του πεδίου latitude βρίσκεται εντός του επιτρεπόμενου εύρους από -90.000 έως 90.000.
- **CHECK (longitude BETWEEN -180.000 AND 180.000)**: Ελέγχει ότι η τιμή του πεδίου longitude βρίσκεται εντός του επιτρεπόμενου εύρους από -180.000 έως 180.000.

Ο πίνακας *festival* δηλώνει τα εξής constraints:

- **UNIQUE(location_id)**: Εξασφαλίζει ότι κάθε location_id εμφανίζεται μόνο μία φορά στον πίνακα, αποτρέποντας την καταχώρηση πολλαπλών φεστιβάλ σε μία τοποθεσία.
- **UNIQUE (year)**: Εξασφαλίζει ότι το πεδίο year, το οποίο υπολογίζεται αυτόματα από το start_date, είναι μοναδικό στον πίνακα.
- **CHECK (end_date >= start_date)**: Ελέγχει ότι η ημερομηνία λήξης (end_date) είναι μετά ή ίση με την ημερομηνία έναρξης (start_date)

Ο πίνακας *event* δηλώνει τα εξής constraints:

- **CHECK (duration BETWEEN 0 AND 720)**: Ελέγχει ότι η διάρκεια του event (σε λεπτά) είναι μεταξύ 0 και 720 λεπτών (δηλαδή, 12 ώρες).

Ο πίνακας *stage* δηλώνει τα εξής constraints:

- **CHECK (max_capacity > 0)**: Ελέγχει ότι η μέγιστη χωρητικότητα (max_capacity) είναι μεγαλύτερη από 0, αποτρέποντας την καταχώριση μη έγκυρης ή αρνητικής τιμής.

Ο πίνακας *staff* δηλώνει τα εξής constraints:

- **CHECK (age > 0)**: Ελέγχει ότι η ηλικία του προσωπικού (age) είναι μεγαλύτερη από το 0.
- **CHECK (experience_level IN ('Specialist','Beginner', 'Intermediate', 'Experienced', 'Very Experienced'))**: Ελέγχει ότι το επίπεδο εμπειρίας (experience_level) ανήκει σε ένα από τα προκαθορισμένα επίπεδα: 'Specialist', 'Beginner', 'Intermediate', 'Experienced', 'Very Experienced'.
- **CHECK (job IN ('technical', 'security', 'assistant'))**: Ελέγχει ότι ο τύπος της εργασίας (job) ανήκει σε ένα από τα προκαθορισμένα επαγγελματικά πεδία: 'technical', 'security', 'assistant'.

Ο πίνακας *stage_equipment* δηλώνει τα εξής constraints:

- **DEFAULT 0**: Ορίζει την προεπιλεγμένη τιμή του πεδίου quantity στο 0, αν δεν καθοριστεί άλλη τιμή κατά την εισαγωγή.
- **CHECK (quantity >= 0)**: Ελέγχει ότι η ποσότητα του εξοπλισμού (quantity) είναι μεγαλύτερη ή ίση με το 0.

Ο πίνακας *artist* δηλώνει τα εξής constraints:

- **CHECK (website LIKE 'https://%')**: Ελέγχει ότι η διεύθυνση ιστοσελίδας (website) αρχίζει με "https://", διασφαλίζοντας την εγκυρότητα του URL.

Ο πίνακας *performance* δηλώνει τα εξής constraints:

- **CHECK (type_of_performance IN ('warm up', 'headline', 'Special guest', 'other'))**: Ελέγχει ότι η τιμή στο πεδίο type_of_performance είναι μία από τις προκαθορισμένες τιμές: 'warm up', 'headline', 'Special guest', ή 'other'.
- **CHECK (duration <= 180)**: Ελέγχει ότι η διάρκεια της εμφάνισης (duration) είναι μικρότερη ή ίση με 180 λεπτά, δηλαδή 3 ώρες.

Ο πίνακας *visitor* δηλώνει τα εξής constraints:

- **CHECK (age > 0)**: Ελέγχει ότι η ηλικία του επισκέπτη είναι μεγαλύτερη από 0, διασφαλίζοντας ότι δεν εισάγεται αρνητική ή μηδενική τιμή για την ηλικία.

Ο πίνακας *visitor_contact* δηλώνει τα εξής constraints:

- **UNIQUE (phone, email)**: Διασφαλίζει ότι η συνδυασμένη τιμή των πεδίων phone και email είναι μοναδική για κάθε εγγραφή επαφής, αποτρέποντας την ύπαρξη διπλών καταχωρήσεων με την ίδια τηλεφωνική γραμμή και διεύθυνση ηλεκτρονικού ταχυδρομείου.

Ο πίνακας *ticket* δηλώνει τα εξής constraints:

- **UNIQUE (EAN13)**: Διασφαλίζει ότι το πεδίο EAN13 (ο αριθμός του εισιτηρίου) είναι μοναδικό για κάθε εγγραφή, αποτρέποντας επανάληψη του ίδιου αριθμού εισιτηρίου.
- **CHECK (cost >= 0)**: Ελέγχει ότι η τιμή του εισιτηρίου (στο πεδίο cost) είναι μεγαλύτερη ή ίση με το 0, εξασφαλίζοντας ότι δεν μπορεί να καταχωρηθεί αρνητική τιμή.
- **CHECK (purchase_method IN ('credit_card', 'debit_card', 'bank_account', 'non_cash', 'cash'))**: Ορίζει τους επιτρεπόμενους τρόπους αγοράς για το εισιτήριο.
- **CHECK (category IN ('general_admission', 'vip', 'backstage', 'other'))**: Ορίζει τις επιτρεπόμενες κατηγορίες εισιτηρίων.
- **DEFAULT TRUE (active)**: Θέτει το πεδίο active στην προεπιλεγμένη τιμή TRUE κατά την εισαγωγή των δεδομένων, δηλώνοντας ότι το εισιτήριο είναι ενεργό από προεπιλογή (ιδιαίτερα για μελλοντικά γεγονότα).
- **UNIQUE (event_id, visitor_id)**: Εξασφαλίζει ότι κάθε επισκέπτης μπορεί να αγοράσει μόνο ένα εισιτήριο για την ίδια παράσταση (μία εγγραφή εισιτηρίου ανά επισκέπτη για κάθε παράσταση).

Ο πίνακας *buyer_interest_queue* περιλαμβάνει τα εξής constraints:

- **CHECK (category IS NULL OR category IN (...))**: Επιτρέπει μόνο έγκυρες κατηγορίες εισιτηρίων ή NULL.
- **DEFAULT FALSE (is_fulfilled)**: Θέτει το πεδίο is_fulfilled στην προεπιλεγμένη τιμή FALSE κατά την εισαγωγή των δεδομένων, δηλώνοντας ότι το αίτημα αγοράς είναι ανικανοποίητο.

Ο πίνακας *resale_queue* περιλαμβάνει τα εξής constraints:

- **DEFAULT FALSE (is_completed)**: Θέτει το πεδίο is_completed στην προεπιλεγμένη τιμή FALSE κατά την εισαγωγή των δεδομένων, δηλώνοντας ότι το αίτημα για πώληση ενός συγκεκριμένου εισιτηρίου δεν έχει ολοκληρωθεί.

Ο πίνακας review περιλαμβάνει τα εξής constraints και περιορισμούς:

- **CHECK (artist_performance BETWEEN 1 AND 5)**: Επιτρέπει μόνο τιμές 1 έως 5 για την απόδοση του καλλιτέχνη.
- **CHECK (sound_lighting BETWEEN 1 AND 5)**: Ελέγχει ότι η βαθμολογία για ήχο και φώτα είναι εντός επιτρεπόμενων ορίων.
- **CHECK (stage_presence BETWEEN 1 AND 5)**: Περιορίζει τη βαθμολογία παρουσίας στη σκηνή σε τιμές 1–5.
- **CHECK (organization BETWEEN 1 AND 5)**: Επιτρέπει μόνο έγκυρες βαθμολογίες για την οργάνωση της εκδήλωσης.
- **CHECK (overall_impression BETWEEN 1 AND 5)**: Ορίζει ότι η συνολική εντύπωση πρέπει να κυμαίνεται μεταξύ 1 και 5.

Επιπλέον constraints

Πρέπει να σημειώσουμε ότι κάθε πίνακας, ο οποίος εμπεριέχει foreign key, το δηλώνει σε constraint με βάση τη σύνταξη:

CONSTRAINT fk_name FOREIGN KEY (column_name)
REFERENCES table_name(column_name) ON DELETE RESTRICT ON UPDATE CASCADE

Αυτό εξασφαλίζει, πως η αναφορά θα γίνεται σε υπαρκτές εγγραφές του σχετιζόμενου πίνακα. Επίσης, απαγορεύονται οι διαγραφές εγγραφών του πίνακα πατέρα (πίνακας στον οποίο αναφέρεται το foreign key), ώστε να μην υπάρξουν «ορφανά παιδιά», ενώ σε τυχόν ενημέρωσή του, να ενημερώνονται και οι τιμές στον πίνακα παιδί.

Τέλος, δεν πρέπει να παραλειφθεί πως η δήλωση ενός attribute ως PRIMARY KEY συνεπάγεται με ένα constraint που επιβάλλει μοναδικότητα (UNIQUE) και NOT NULL τιμές.

Triggers/Procedures

Οι περίπλοκοι περιορισμοί της εκφώνησης δεν ήταν εφικτό να εφαρμοστούν αποκλειστικά με τα constraints κατά τη δημιουργία του πίνακα, καθώς αυτά δεν υποστηρίζουν πολύπλοκους υπολογισμούς. Για τον λόγο αυτό, σχεδιάστηκαν πολλαπλά triggers-procedures, τα οποία διασφαλίζουν την εισαγωγή και τη διατήρηση στη βάση ορθής πληροφορίας ευθυγραμμισμένης με τις απαιτήσεις της άσκησης.

Ακολουθεί μία σύντομη περιγραφή των triggers-procedures, ο κώδικας των οποίων περικλείεται στο αρχείο triggers_procedures.sql και παραλείπεται λόγω συντομίας της αναφοράς.

Trigger 1 / check_event_overlap

Το trigger check_event_overlap ενεργοποιείται πριν την εισαγωγή ενός νέου event στον πίνακα event και εκτελεί μια σειρά από ελέγχους που εξασφαλίζουν τη χρονική συνοχή και τη συμμόρφωση με τους κανόνες του φεστιβάλ. Αρχικά, ελέγχει ότι οι ημερομηνίες και ώρες του event βρίσκονται εντός του εύρους ημερομηνιών του αντίστοιχου φεστιβάλ (festival.start_date έως festival.end_date). Στη συνέχεια, επιβεβαιώνει ότι δεν υπάρχει ήδη άλλο event που να πραγματοποιείται στην ίδια σκηνή (stage_id) και να επικαλύπτεται χρονικά με το νέο. Τέλος, εξετάζει αν η προσθήκη του event παραβιάζει τη μέγιστη ημερήσια διάρκεια φεστιβάλ για τη συγκεκριμένη ημέρα — η συνολική διάρκεια όλων των events ανά ημέρα δεν πρέπει να υπερβαίνει τα 780 λεπτά (δηλαδή 13 ώρες). Αν παραβιαστεί οποιοσδήποτε από αυτούς τους περιορισμούς, η εισαγωγή απορρίπτεται με κατάλληλο μήνυμα σφάλματος.

Trigger 2 / check_staff_assignments

Το trigger check_staff_assignments ενεργοποιείται πριν την εισαγωγή μιας νέας ανάθεσης προσωπικού στον πίνακα staff_event και διασφαλίζει την εγκυρότητα της συσχέτισης μεταξύ εργαζομένου και event. Αρχικά, εντοπίζει τον ρόλο του υπαλλήλου από τον πίνακα staff και τις χρονικές πληροφορίες του αντίστοιχου event από τον πίνακα event. Στη συνέχεια, ελέγχει αν ο υπάλληλος είναι ήδη ανατεθειμένος σε κάποιο άλλο event που επικαλύπτεται χρονικά με το νέο, ώστε να αποτρέπεται η ταυτόχρονη συμμετοχή σε πολλαπλές

εκδηλώσεις. Επιπλέον, αν ο υπάλληλος ανήκει στο τεχνικό προσωπικό, γίνεται έλεγχος για το πλήθος των αναθέσεων μέσα στην ίδια ημέρα — δεν επιτρέπεται να συμμετέχει σε περισσότερα από δύο events ημερησίως. Αν παραβιαστεί κάποιος από τους παραπάνω κανόνες, η εισαγωγή μπλοκάρεται με μήνυμα σφάλματος, εξασφαλίζοντας έτσι την ορθολογική κατανομή του προσωπικού και την αποφυγή υπερφόρτωσης.

Procedure 1 / validate_event_staffing

Το procedure validate_event_staffing ελέγχει τη σωστή στελέχωση προσωπικού για κάθε event, βάσει προκαθορισμένων ποσοστιαίων ορίων ανά ειδικότητα (job). Για κάθε event, γίνεται σύγκριση του αριθμού του προσωπικού με τη μέγιστη χωρητικότητα της σκηνής στην οποία λαμβάνει χώρα, εξασφαλίζοντας ότι υπάρχουν τουλάχιστον 1 τεχνικός, τουλάχιστον 5% του μέγιστου κοινού σε άτομα ασφαλείας, και τουλάχιστον 2% σε βοηθητικό προσωπικό. Αν για οποιοδήποτε event δεν ικανοποιούνται αυτοί οι ελάχιστοι αριθμοί, παράγεται σφάλμα που μπλοκάρει την εκτέλεση και επιστρέφει σχετικό μήνυμα. Με αυτόν τον τρόπο διασφαλίζεται ότι κάθε event έχει την ελάχιστη απαραίτητη κάλυψη σε κρίσιμες θέσεις εργασίας για την ομαλή και ασφαλή διεξαγωγή του.

Trigger 3 / trg_prevent_event_deletion

Το trigger trg_prevent_event_deletion ενεργοποιείται πριν από κάθε απόπειρα διαγραφής μιας εγγραφής από τον πίνακα event και έχει ως αποκλειστικό σκοπό την πλήρη απαγόρευση διαγραφών. Χωρίς να κάνει άλλους ελέγχους, εκτελεί άμεσα εντολή SIGNAL που παράγει μήνυμα σφάλματος και μπλοκάρει τη λειτουργία, με σαφές μήνυμα ότι τα events δεν μπορούν να διαγραφούν. Το trigger αυτό μοιάζει πλεονάζον, καθώς το primary key του event αποτελεί foreign key σε άλλους πίνακες και η διαγραφή του αποτρέπεται από τα constraints των foreign keys λόγω του ON DELETE RESTRICT.

Trigger 4 / trg_prevent_festival_deletion

Το trigger αυτό εκτελεί την ίδια λειτουργία με το προηγούμενο trigger αποτρέποντας τη διαγραφή κάποιου festival.

Trigger 5 / trg_check_performance_gaps

Το trigger trg_check_performance_gaps ενεργοποιείται πριν την εισαγωγή μιας νέας εμφάνισης στον πίνακα performance και εξασφαλίζει τη χρονική συνέπεια και κανονικότητα του προγράμματος κάθε event. Πρώτα, ελέγχει ότι η χρονική διάρκεια της εμφάνισης βρίσκεται εξολοκλήρου εντός του διαστήματος του αντίστοιχου event. Έπειτα, διασφαλίζεται ότι η νέα εμφάνιση δεν επικαλύπτεται με καμία ήδη καταχωρισμένη στο ίδιο event. Εξετάζεται επίσης αν το κενό μεταξύ εμφανίσεων είναι τουλάχιστον 5 λεπτά. Τέλος, απορρίπτεται η εισαγωγή αν δημιουργείται κενό μεγαλύτερο των 30 λεπτών πριν ή μετά την εμφάνιση. Αν παραβιαστεί οποιαδήποτε από αυτές τις συνθήκες, η εισαγωγή μπλοκάρεται με σχετικό μήνυμα σφάλματος.

Trigger 6 / trg_check_artist_band_constraints

Το trigger trg_check_artist_band_constraints ενεργοποιείται πριν από κάθε εισαγωγή στον πίνακα performance_artist και έχει ως στόχο τη συμμόρφωση με σημαντικούς λειτουργικούς περιορισμούς. Αρχικά, αποτρέπει την εισαγωγή διπλότυπης εγγραφής για τον ίδιο καλλιτέχνη σε ίδια εμφάνιση, ένας περιορισμός που θα γινότανε σε δεύτερο χρόνο από τον περιορισμό του PRIMARY KEY (artist_id,performance_id), αλλά προτιμάται να ελεγχθεί εδώ, για να διασφαλιστεί η σωστή σειρά των ελέγχων. Στη συνέχεια, ελέγχει αν ο καλλιτέχνης είναι ήδη προγραμματισμένος να εμφανιστεί σε άλλη σκηνή την ίδια χρονική στιγμή, αποτρέποντας έτσι την ταυτόχρονη παρουσία σε πολλαπλές σκηνές. Επιπλέον, το trigger εξετάζει το ιστορικό συμμετοχών του καλλιτέχνη και υπολογίζει αν η νέα συμμετοχή του θα δημιουργούσε μια ακολουθία άνω των τριών συνεχόμενων ετών παρουσίας σε φεστιβάλ. Αν βρεθεί ότι ο καλλιτέχνης έχει ή θα έχει συμμετάσχει για περισσότερα από τρία διαδοχικά έτη, η εισαγωγή απορρίπτεται. Όλοι οι έλεγχοι υποστηρίζονται με κατάλληλα μηνύματα σφάλματος, διασφαλίζοντας την επιχειρησιακή λογική και την ποιότητα του προγράμματος.

*Η εισαγωγή μίας εγγραφής πρώτα ελέγχεται από το trigger BEFORE INSERT ON και έπειτα φιλτράρεται ως προς το constraint UNIQUE που επιβάλλει το PRIMARY KEY του συγκεκριμένου πίνακα.

Trigger 7 / check_stage_capacity

Το trigger check_stage_capacity ενεργοποιείται πριν από κάθε εισαγωγή εισιτηρίου στον πίνακα ticket και εξασφαλίζει ότι η διαδικασία αγοράς σέβεται μια σειρά από κρίσιμους περιορισμούς. Αρχικά ελέγχει αν η αγορά εισιτηρίου γίνεται μετά τη λήξη του event και, αν ναι, την απορρίπτει. Στη συνέχεια, ελέγχει την ενεργή κατάσταση του εισιτηρίου και αποτρέπει τη δημιουργία ανενεργού εισιτηρίου πριν από την έναρξη του event. Επιπλέον, συγκρίνει τον αριθμό εισιτηρίων που έχουν ήδη εκδοθεί για το συγκεκριμένο event με τη μέγιστη χωρητικότητα της σκηνής και εμποδίζει την υπέρβασή της. Τέλος, για τις VIP κατηγορίες εισιτηρίων, διασφαλίζει ότι δεν ξεπερνούν το 10% της συνολικής χωρητικότητας της σκηνής, εφαρμόζοντας σχετικό όριο. Σε κάθε παραβίαση των παραπάνω κανόνων, το trigger παράγει κατανοητά μηνύματα σφάλματος, διατηρώντας έτσι την αξιοπιστία και την ομαλή λειτουργία του συστήματος πωλήσεων.

Trigger 8 / new_resale_queue

Το trigger new_resale_queue ενεργοποιείται πριν από κάθε εισαγωγή στη λίστα μεταπώλησης (resale_queue) και εφαρμόζει αυστηρούς ελέγχους για την εγκυρότητα της συναλλαγής. Αρχικά, διασφαλίζει ότι ο επισκέπτης που προσπαθεί να πουλήσει το εισιτήριο είναι ο πραγματικός κάτοχός του. Στη συνέχεια, ελέγχει εάν έχει καλυφθεί η μέγιστη χωρητικότητα της σκηνής για να είναι επιτρεπτή η μεταπώληση — δηλαδή, η μεταπώληση επιτρέπεται μόνο όταν το event είναι sold out. Επίσης, το trigger αποτρέπει την είσοδο εισιτηρίων στη λίστα μεταπώλησης αν το event έχει ήδη λήξει ή αν η ώρα αιτήματος για μεταπώληση προηγείται της ώρας αγοράς του εισιτηρίου, με το οποίο εξαντλείται η χωρητικότητα της σκηνής. Τέλος, ελέγχει εάν το εισιτήριο είναι ακόμα ενεργό, απορρίπτοντας την καταχώριση αν πρόκειται για ήδη χρησιμοποιημένο εισιτήριο. Με αυτόν τον τρόπο, το trigger προστατεύει την αξιοπιστία της διαδικασίας μεταπώλησης, εξασφαλίζοντας ότι γίνεται μόνο από

νόμιμους κατόχους, σε κατάλληλες συνθήκες και εντός των σωστών χρονικών πλαισίων.

Procedure 2 / MatchResaler

Η αποθηκευμένη διαδικασία MatchResaler καλείται μόνο από τη διαδικασία my_insert_resaler μετά την εισαγωγή ενός πωλητή, ελέγχοντας αν υπάρχει κάποιος συμβατός ενδιαφερόμενος αγοραστής σε εκκρεμότητα στη λίστα buyer_interest_queue. Αρχικά, ανακτά το event_id και την category του εισιτηρίου προς μεταπώληση. Στη συνέχεια, εντοπίζει τον πρώτο διαθέσιμο ενδιαφερόμενο αγοραστή (με βάση τη σειρά προτεραιότητας request_time) που είτε έχει ζητήσει συγκεκριμένο ticket_id, είτε εισιτήριο του ίδιου event και κατηγορίας. Αν ο ενδιαφερόμενος δεν είναι ήδη εγγεγραμμένος ως επισκέπτης (visitor_id IS NULL), προστίθεται στον πίνακα visitor με τα στοιχεία του (buyer_name, age). Το εισιτήριο μεταβιβάζεται στον νέο ή υφιστάμενο επισκέπτη και η μεταπώληση ολοκληρώνεται με ενημέρωση των resale_queue και buyer_interest_queue, μαρκάροντας τις σχετικές εγγραφές ως ολοκληρωμένες (is_completed = TRUE, is_fulfilled = TRUE). Η διαδικασία αυτοματοποιεί και διασφαλίζει την άμεση αντιστοίχιση ανάμεσα σε πωλητές και αγοραστές, ενισχύοντας την ομαλή λειτουργία της αγοράς μεταπωλήσεων.

Procedure 3 / my_insert_resaler

Η αποθηκευμένη διαδικασία my_insert_saler χρησιμοποιείται για την εισαγωγή ενός εισιτηρίου προς μεταπώληση στον πίνακα resale_queue, καθώς και για την αυτόματη προσπάθεια αντιστοίχισής του με κάποιον διαθέσιμο αγοραστή. Δέχεται ως παραμέτρους το ticket_id, το visitor_id του πωλητή και την ημερομηνία/ώρα του αιτήματος (request_time). Αρχικά, καταχωρεί τα δεδομένα στον πίνακα resale_queue και στη συνέχεια, μέσω της εντολής SELECT LAST_INSERT_ID(), ανακτά το αναγνωριστικό της νέας εγγραφής. Αμέσως μετά καλεί τη διαδικασία MatchResaler, η οποία ελέγχει αν υπάρχει κάποιος αγοραστής σε εκκρεμότητα που να ταιριάζει με το εισιτήριο, ώστε να προχωρήσει άμεσα η μεταπώληση. Αν υπάρχουν περισσότεροι από έναν αγοραστές, τότε επιλέγεται αυτός με το πιο παλιό request_time. Έτσι, η my_insert_resaler λειτουργεί ως "entry

point" για τη μεταπώληση, συνδυάζοντας την καταχώριση στην ουρά, η οποία λειτουργεί με σειρά FIFO, και τη δυναμική αντιστοίχιση σε μία ενιαία διαδικασία.

Trigger 9 / new_buyer

Το trigger new_buyer ενεργοποιείται πριν από κάθε εισαγωγή στον πίνακα buyer_interest_queue και ελέγχει την εγκυρότητα των στοιχείων του αγοραστή και τη δυνατότητα αγοράς εισιτηρίου. Αρχικά, επιβεβαιώνει ότι έχει δηλωθεί είτε συγκεκριμένο εισιτήριο (ticket_id) είτε συνδυασμός παράστασης και κατηγορίας (event_id και category). Αν έχει δηλωθεί visitor_id, επαληθεύεται ότι αντιστοιχεί στο όνομα του αγοραστή. Ύστερα, ελέγχεται αν ο αγοραστής ήδη κατέχει εισιτήριο για το ίδιο event και στην περίπτωση που δηλωθεί ticket_id, αν το εισιτήριο είναι πράγματι διαθέσιμο προς πώληση μέσω του resale_queue. Επιπλέον, ελέγχεται αν έχει καλυφθεί η χωρητικότητα του χώρου (stage), αφού μόνο τότε επιτρέπεται η ένταξη αγοραστών στη λίστα αναμονής μεταπώλησης. Τέλος, εξετάζεται ότι το αίτημα γίνεται πριν τη λήξη του event και μετά την αγορά του τελευταίου εισιτηρίου, ώστε να διασφαλίζεται χρονική συνέπεια στη διαδικασία αγοράς ήδη διατεθειμένων εισιτηρίων.

Procedure 3 / MatchBuyer

Η αποθηκευμένη διαδικασία MatchBuyer καλείται μόνο από τη διαδικασία my_insert_buyer_interest_queue μετά την εισαγωγή ενός αγοραστή ελέγχοντας αν υπάρχει κάποιος συμβατός πωλητής σε εκκρεμότητα στη λίστα resale_queue. Αν ο αγοραστής έχει δηλώσει συγκεκριμένο ticket_id, τότε γίνεται αναζήτηση για αυτό το εισιτήριο. Διαφορετικά, αναζητείται εισιτήριο που αντιστοιχεί στο ίδιο event_id και category. Αν βρεθεί αντιστοιχία, δημιουργείται εγγραφή στον πίνακα visitor εφόσον δεν υπάρχει ήδη, ενημερώνεται ο κάτοχος του εισιτηρίου και σημειώνονται ως ολοκληρωμένες οι αντίστοιχες εγγραφές στους πίνακες resale_queue και buyer_interest_queue.

Procedure 4 / my_insert_buyer_interest_queue

Η αποθηκευμένη διαδικασία my_insert_buyer_interest_queue εισάγει έναν νέο αγοραστή στον πίνακα buyer_interest_queue και αμέσως καλεί τη διαδικασία MatchBuyer για να ελέγξει αν υπάρχει διαθέσιμος πωλητής με εισιτήριο που να ταιριάζει είτε με το ticket_id, είτε με τον συνδυασμό event_id και category. Αν υπάρχουν περισσότεροι από έναν πωλητές, τότε επιλέγεται αυτός με το πιο παλιό request_time. Με τον εντοπισμό κατάλληλου πωλητή, ολοκληρώνεται αυτόματα η μεταπώληση του εισιτηρίου. Έτσι η διαδικασία αυτή σε συνδυασμό με την MatchBuyer καταφέρνουν να εφαρμόσουν την απαίτηση για FIFO σειρά μεταπώλησης.

Trigger 10 / valid_review

Το trigger valid_review ενεργοποιείται πριν από κάθε εισαγωγή στη σχέση review και ελέγχει αν ο επισκέπτης που επιχειρεί να κάνει την αξιολόγηση έχει παρακολουθήσει την παράσταση. Συγκεκριμένα, ανακτά το event_id που αντιστοιχεί στην performance_id της κριτικής και στη συνέχεια επιβεβαιώνει ότι ο επισκέπτης (visitor_id) διαθέτει ανενεργό εισιτήριο (active = FALSE) για το συγκεκριμένο event. Αν δεν πληρείται αυτή η προϋπόθεση, απορρίπτεται η εισαγωγή με μήνυμα σφάλματος 'Invalid review'.

Ευρετήρια-Indexes

Κατά τον ορισμό των ευρετηρίων (indexes) στη βάση δεδομένων λαμβάνεται υπόψη ότι για κάθε πίνακα δημιουργούνται αυτόματα ευρετήρια για τα πεδία που αποτελούν primary keys. Αυτό είναι κρίσιμο, καθώς τα primary keys χρησιμοποιούνται πολύ συχνά σε WHERE συνθήκες, JOIN εντολές και triggers που εφαρμόζουν περιορισμούς σε πολλούς πίνακες. Συνεπώς, η ύπαρξη ευρετηρίων σε αυτά τα πεδία εξασφαλίζει ταχύτατη πρόσβαση σε εγγραφές.

Επιπλέον, ευρετήρια δημιουργούνται και για foreign keys που δεν είναι primary keys καθώς είναι σημαντικό να μπορεί η βάση να ελέγξει γρήγορα τις σχέσεις αναφοράς, π.χ. κατά τη διαγραφή ή ενημέρωση εγγραφών που σχετίζονται με άλλους πίνακες.

Αξίζει, ακόμα, να αναφερθούμε στα composite indexes, τα οποία συνιστούν σύνθετα ευρετήρια. Όταν ορίζεται composite index πάνω στα πεδία (attribute1, attribute2, attribute3), τότε το ίδιο ευρετήριο μπορεί να χρησιμοποιηθεί για αναζητήσεις που βασίζονται σε οποιοδήποτε υποσύνολο-πρόθεμα της σειράς αυτής, όπως (attribute1), (attribute1, attribute2). Ωστόσο, για να χρησιμοποιηθεί το ευρετήριο σε άλλα υποσύνολα των πεδίων (π.χ (attribute2, attribute3)), απαιτείται ξεχωριστό ευρετήριο.

Συνάμα, indexes δημιουργούνται σε πεδία που σχετίζονται με ημερομηνίες έναρξης και λήξης σε πίνακες όπως festival, event, performance, εφόσον αυτά τα πεδία εμφανίζονται συχνά σε ερωτήματα/triggers/procedures.

Πρέπει να σημειωθεί ότι όταν εφαρμόζεται UNIQUE περιορισμός σε κάποια πεδία, δημιουργείται αυτόματα ευρετήριο γι’ αυτά. Άρα, δεν είναι απαραίτητο να δημιουργηθεί επιπλέον index για λόγους αναζήτησης.

Τέλος, σημειώνεται ότι όλα τα indexes έχουν σχεδιαστεί κατά προσέγγιση, με σκοπό τη βελτίωση της απόδοσης της βάσης. Παρακάτω περιγράφονται πιο αναλυτικά ορισμένες ξεχωριστές περιπτώσεις ευρετηρίων που δεν περιλήφθηκαν στην άνωθι γενική περιγραφή:

- (festival_id, festival_name) στον πίνακα festival για το αποδοτικότερο GROUP BY σε πολλαπλά queries. Ενδεικτικά στα queries 7, 10, 12.
- (job) στον πίνακα staff για την αποδοτικότερη δημιουργία του προσωρινού πίνακα temp1 (WHERE clause).
- (artist_id, artist_name, pseudonym) στον πίνακα artist για το ταχύτερο GROUP BY σε queries. Ενδεικτικά το ευρετήριο αυτό αξιοποιείται στα queries 3, 5.
- (artist_name) στον πίνακα artist για την αποδοτική εκτέλεση του WHERE clause στο query 4. Πιο αναλυτικά η χρησιμότητα του εξηγείται παρακάτω.
- (type_of_performance) στον πίνακα performance ο οποίος βελτιώνει την απόδοση του query 3 στην δημιουργία του προσωρινού πίνακα warmup_performances και συγκεκριμένα κατά την εξέταση του WHERE clause. Για τον ίδιο πίνακα τα ευρετήρια στα attributes (start_time) και (end_time) βοηθούν στην γρήγορη υλοποίηση ορισμένων triggers όπως το 6.
- (start_time) στον πίνακα event ο οποίος χρησιμοποιείται στο WHERE clause για την δημιουργία του temporary πίνακα occupied_assistant_staff στο query 8. Αυτό και το index στο (end_time) είναι ευρετήρια τα οποία χρησιμοποιούνται σε πολλαπλά WHERE clauses σε triggers, όπως για παράδειγμα στο trigger 1.
- (genre) στον πίνακα artist_genre το οποίο χρησιμοποιείται για τη βελτίωση της σύνθεσης του προσωρινού πίνακα artists_in_genre στο query 2.
- (visitor_name) στον πίνακα visitor το οποίο συμβάλλει στο βέλτιστο query plan για την επίλυση του query 6, επιτυγχάνοντας ένα ταχύ WHERE clause.
- (visitor_id, visitor_name) στον πίνακα visitor το οποίο βοηθά για γρήγορα GROUP BY σε queries όπως τα queries 6 και 9.
- (request_time) στους πίνακες resale_queue,buyer_interest_queue τα οποία επιταχύνουν την ταξινόμηση που εφαρμόζει το ORDER BY προκειμένου τα triggers/procedure να υλοποιούν την FIFO ουρά αγοραπωλησίας.

Inserts

Στο αρχείο «load.sql» υπάρχουν inserts εντολές για κάθε πίνακα του σχήματος. Οι εγγραφές αυτές δημιουργήθηκαν με τέτοιο τρόπο, ώστε να μην παραβιάζουν τους περιορισμούς που θέτει η άσκηση και να μην ενεργοποιούν κανένα trigger ή constraint που σχεδιάσαμε. Τα περισσότερα από αυτά παρήχθησαν με τη βοήθεια τεχνητής νοημοσύνης. Οι εισαγωγές, ωστόσο, στους πίνακες staff_event και performance_artist που έπρεπε να υπακούουν σε περίπλοκους περιορισμούς δημιουργήθηκαν μέσω python scripts, τα οποία βρίσκονται στα αρχεία «code/staff_event_insert.sql» και «code/performance_artist_insert.sql» αντίστοιχα.

Αντίθετα, στο αρχείο «inserts_for_triggers.sql» εμπεριέχονται συγκεκριμένα inserts τα οποία έχουν σχεδιαστεί, ώστε να παραβιάζουν τους περιορισμούς και να ενεργοποιούν τα αντίστοιχα triggers, με σκοπό τη διευκόλυνση του ελέγχου τους. Ενδεικτικά παραθέτουμε τα αποτελέσματα κάποιων τέτοιων εισαγωγών.

```
MariaDB [pulse_uni_schema]> --TRIGGER 1/υπερβαίνει το ημερήσιο όριο του φεστιβάλ σε διάρκεια
MariaDB [pulse_uni_schema]> INSERT INTO event (festival_id, stage_id, event_name, start_time, end_time) VALUES
    -> (1, 30, 'test_event', '2016-06-10 23:30:00', '2016-06-10 23:40:00');
ERROR 1644 (45000): Violation of festival duration per day
```

```
MariaDB [pulse_uni_schema]> --TRIGGER 2/ταυτόχρονη παρουσία προσωπικού
MariaDB [pulse_uni_schema]> INSERT INTO event (festival_id, stage_id, event_name, start_time, end_time) VALUES      #προσθήκη event την ίδια ωρα#
    -> (1, 30, 'test_event', '2016-06-10 9', '2016-06-10 10');
Query OK, 1 row affected (0.007 sec)

MariaDB [pulse_uni_schema]> INSERT INTO staff_event (staff_id, event_id) VALUES
    -> (1, 51);
ERROR 1644 (45000): One person cannot be in two events at the same time. Overlap assignments
MariaDB [pulse_uni_schema]> --PROCEDURE 1/Υποστελεχωμένο event
MariaDB [pulse_uni_schema]> CALL validate_event_staffing();
ERROR 1644 (45000): Staffing constraints violated for one or more events
```

```
MariaDB [pulse_uni_schema]> --TRIGGER 3/Break μικρότερο από 5 minutes
MariaDB [pulse_uni_schema]> INSERT INTO performance (event_id, type_of_performance, start_time, end_time) VALUES
    -> (1, 'Warm up', '2016-06-10 9:32:00', '2016-06-10 9:35:00');
ERROR 1644 (45000): The gap between performances must be at least 5 minutes
```

```
MariaDB [pulse_uni_schema]> --αυτο θα σκασει γιατι ξεπερνα τα 3 συνεχομενα χρονια συμμετοχης
MariaDB [pulse_uni_schema]> INSERT INTO performance_artist(performance_id, artist_id) VALUES (41, 30);
ERROR 1644 (45000): Artist cannot participate for more than 3 consecutive years
```

Queries

Query 1:

```sql
#------------------QUERY 1----------------#
SELECT
    YEAR(e.start_time) AS festival_year,
    SUM(CASE WHEN t.purchase_method = 'cash' THEN t.cost ELSE 0 END) AS cash,
    SUM(CASE WHEN t.purchase_method = 'debit_card' THEN t.cost ELSE 0 END) AS debit_card,
    SUM(CASE WHEN t.purchase_method = 'credit_card' THEN t.cost ELSE 0 END) AS credit_card,
    SUM(CASE WHEN t.purchase_method = 'bank_account' THEN t.cost ELSE 0 END) AS bank_account,
    SUM(CASE WHEN t.purchase_method = 'non_cash' THEN t.cost ELSE 0 END) AS non_cash,
    SUM(t.cost) AS total_earnings
FROM ticket t
JOIN event e ON t.event_id = e.event_id
GROUP BY festival_year
ORDER BY festival_year;
```

- Για κάθε έτος του φεστιβάλ (GROUP BY festival_year) διαχωρίζουμε τα έσοδα από την πώληση των εισιτηρίων ανάλογα με τον τρόπο πληρωμής με την βοήθεια του CASE WHEN .. END και τα αθροίζουμε χρησιμοποιώντας τη συνάρτηση SUM.

Query 2:

```sql
#------------------QUERY 2----------------#
DELIMITER //
DROP PROCEDURE IF EXISTS get_artists_by_genre_and_year_cte;
CREATE PROCEDURE get_artists_by_genre_and_year_cte (
    IN in_festival_year INT,
    IN in_genre VARCHAR(50)
)
BEGIN
    WITH artists_in_genre AS (
        SELECT
            a.artist_id,
            a.artist_name,
            a.pseudonym,
            ag.genre,
            ag.subgenre
        FROM artist a
        JOIN artist_genre ag ON a.artist_id = ag.artist_id
        WHERE ag.genre = in_genre
    ),
    artists_participated AS (
        SELECT DISTINCT pa.artist_id
        FROM performance_artist pa
        JOIN performance p ON pa.performance_id = p.performance_id
        WHERE YEAR(p.start_time) = in_festival_year
    )
    SELECT
        ag.artist_id,
        ag.artist_name,
        ag.pseudonym,
        ag.genre,
        ag.subgenre,
        CASE
            WHEN ap.artist_id IS NOT NULL THEN 'YES'
            ELSE 'NO'
        END AS participated_in_year
    FROM artists_in_genre ag
    LEFT JOIN artists_participated ap ON ag.artist_id = ap.artist_id
    ORDER BY participated_in_year DESC, ag.artist_id;
END //
DELIMITER ;
```

- Το ερώτημα 2 απαντήθηκε με την μορφή διαδικασίας η οποία παίρνει σαν ορίσματα το έτος και το μουσικό είδος που μας ενδιαφέρουν κάθε φορά. Χρησιμοποιούνται δύο CTEs (Common Table Expressions): ένας για την εύρεση των καλλιτεχνών που ανήκουν στο είδος, και ένας για όσους συμμετείχαν σε παραστάσεις τη συγκεκριμένη χρονιά. Η τελική έξοδος περιλαμβάνει όλους τους καλλιτέχνες του συγκεκριμένου είδους είτε συμμετείχαν το εν λόγω έτος είτε όχι (LEFT JOIN), τα βασικά στοιχεία των καλλιτεχνών μαζί με μία ένδειξη (YES/NO) για τη συμμετοχή τους (CASE).

Query 3

```sql
#------------------QUERY 3---------------#
WITH warmup_performances AS (
    SELECT
        pa.artist_id,
        a.artist_name,
        a.pseudonym,
        YEAR(p.start_time) AS festival_year,
        COUNT(*) AS warmup_count
    FROM artist a
    JOIN performance_artist pa ON a.artist_id = pa.artist_id
    JOIN performance p FORCE INDEX (idx_type_of_performance) ON pa.performance_id = p.performance_id
    WHERE p.type_of_performance = 'warm up'
    GROUP BY a.artist_id, a.artist_name, a.pseudonym, YEAR(p.start_time)
),
warmup_filtered AS (
    SELECT *
    FROM warmup_performances
    WHERE warmup_count > 2
)
SELECT *
FROM warmup_filtered
ORDER BY festival_year, warmup_count DESC;
```

- Στο πρώτο CTE (warmup_performances) υπολογίζεται για κάθε καλλιτέχνη και ανά έτος πόσες φορές συμμετείχε σε warm up εμφανίσεις. Το δεύτερο CTE (warmup_filtered) φιλτράρει όσους συμμετείχαν περισσότερες από δύο φορές. Η τελική έξοδος εμφανίζει τα αποτελέσματα ταξινομημένα ανά έτος και φθίνουσα σειρά συμμετοχών.

Query 4:

```
#------------------QUERY 4---------------#
DELIMITER //
DROP PROCEDURE IF EXISTS avg_review_of_artist_1;
CREATE PROCEDURE avg_review_of_artist_1 (
    IN in_artist_name VARCHAR(100)
)
BEGIN
SELECT a.artist_id,
a.artist_name,
a.pseudonym,
AVG(r.artist_performance) AS avg_artist_performance_review,
AVG(r.overall_impression) AS avg_overall_impression_review
FROM review r
JOIN performance_artist pa ON r.performance_id = pa.performance_id
JOIN artist a IGNORE INDEX (idx_artist_name) ON pa.artist_id = a.artist_id
WHERE a.artist_name = in_artist_name;
END//
DELIMITER ;
```

- Το ερώτημα αυτό απαντήθηκε μέσω της διαδικασίας avg_review_of_artist_1, ώστε ο χρήστης να δίνει σαν όρισμα έναν συγκεκριμένο καλλιτέχνη. Η διαδικασία επιστρέφει τους μέσους όρους αξιολογήσεων του καλλιτέχνη, βάσει του ονόματός του (artist_name). Συγκεκριμένα, υπολογίζονται οι μέσες τιμές για την απόδοσή του σε εμφανίσεις και για τη συνολική εντύπωση που άφησε. Η αναζήτηση γίνεται μέσω συσχετίσεων στους πίνακες review, performance_artist και artist, ενώ παρακάμπτεται σκόπιμα το ευρετήριο στο artist_name με IGNORE INDEX.

Query 4/Εναλλακτικό Query Plan:

```
#-----------with FORCE INDEX-----------#
DELIMITER //
DROP PROCEDURE IF EXISTS avg_review_of_artist_2;
CREATE PROCEDURE avg_review_of_artist_2 (
    IN in_artist_name VARCHAR(100)
)
BEGIN
EXPLAIN SELECT a.artist_id,
a.artist_name,
a.pseudonym,
AVG(r.artist_performance) AS avg_artist_performance_review,
AVG(r.overall_impression) AS avg_overall_impression_review
FROM review r
JOIN performance_artist pa ON r.performance_id = pa.performance_id
JOIN artist a FORCE INDEX (idx_artist_name) ON pa.artist_id = a.artist_id
WHERE a.artist_name = in_artist_name;
END//
DELIMITER ;
```

Στην υλοποίηση επαναλαμβάνεται ο κώδικας και η λογική του προηγούμενου πλάνου με την διαφορά ότι επιβάλλουμε στον πίνακα artist να χρησιμοποιηθεί ως key το ευρετήριο idx_artist_name. Η τροποποίηση αυτή προκαλεί αλλαγή στη σειρά εκτέλεσης του JOIN. Στην αρχή κρατούνται οι εγγραφές του πίνακα artist με artist_name = in_artist_name και έπειτα γίνονται τα JOINS με τους υπόλοιπους πίνακες.

Συμπεραίνουμε πως η επιλογή του σωστού ευρετηρίου καθιστά δυνατή την βέλτιστη εκτέλεση του WHERE clause στον πίνακα αναφοράς (artist) χωρίς να απαιτείται η προσπέλαση όλων των γραμμών του πίνακα. Έτσι προκύπτει ένας πίνακας με μικρό αριθμό γραμμών (συνήθως) ο οποίος τοποθετείται πρώτος στην σειρά των JOIN και συνδέεται αποδοτικά με τους υπόλοιπους. Αντίθετα χωρίς την υπόδειξη (ύπαρξη) του κατάλληλου index αυτό το «φιλτράρισμα» δεν θα ήταν αποδοτικό καθώς το WHERE θα έπρεπε να διατρέξει όλες τις γραμμές του πίνακα. Γι' αυτό εκτελούνται τα JOIN με την σειρά που δόθηκαν και στο αποτέλεσμα εφαρμόζεται η συνθήκη του WHERE.

Η βελτίωση της επίδοσης φαίνεται τόσο στο χρόνο που εκτελείται το query όσο και στα rows που γίνονται "access", όπως φανερώνει το σχετικό trace με τη βοήθεια της εντολής EXPLAIN.

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	r	ALL	idx_fk_performance_id	NULL	NULL	NULL	24	
1	SIMPLE	pa	ref	PRIMARY,idx_fk_performance_id,idx_fk_artist_id,idx_performance_artist_ids	idx_fk_performance_id	4	pulse_uni_schema.r.performance_id	2	Using index
1	SIMPLE	a	eq_ref	PRIMARY	PRIMARY	4	pulse_uni_schema.pa.artist_id	1	Using where

key
NULL
idx_fk_performance_id
PRIMARY

rows	Extra
24	
2	Using index
1	Using where

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	a	ref	idx_artist_name	idx_artist_name	402	const	1	Using index condition
1	SIMPLE	pa	ref	PRIMARY,idx_fk_performance_id,idx_fk_artist_id,idx_performance_artist_ids	idx_fk_artist_id	4	pulse_uni_schema.a.artist_id	5	Using index
1	SIMPLE	r	ref	idx_fk_performance_id	idx_fk_performance_id	4	pulse_uni_schema.pa.performance_id	1	

key
idx_artist_name
idx_fk_artist_id
idx_fk_performance_id

rows	Extra
1	Using index condition
5	Using index
1	

Query 5:

```sql
#-------------------QUERY 5----------------#
SELECT
a.artist_id,
a.artist_name,
a.pseudonym,
TIMESTAMPDIFF(YEAR, a.birth_date, NOW()) AS age,
COUNT(DISTINCT YEAR(p.start_time)) AS total_festival_participations
FROM artist a
JOIN performance_artist pa ON a.artist_id = pa.artist_id
JOIN performance p ON pa.performance_id = p.performance_id
WHERE TIMESTAMPDIFF(YEAR, a.birth_date, NOW()) < 30
GROUP BY a.artist_id, a.artist_name, a.pseudonym
ORDER BY total_festival_participations DESC, age;
```

- Το ερώτημα επιστρέφει καλλιτέχνες κάτω των 30 ετών (βάσει της σημερινής ημερομηνίας – χρήση TIMESTAMPDIFF και NOW()) και πληροφορίες σχετικά με τη συμμετοχή τους σε φεστιβάλ. Συγκεκριμένα, υπολογίζει την ηλικία τους και τον συνολικό αριθμό διαφορετικών (DISTINCT) ετών συμμετοχής σε φεστιβάλ Η έξοδος ομαδοποιείται ανά καλλιτέχνη και ταξινομείται κατά αριθμό συμμετοχών σε φθίνουσα σειρά, και στη συνέχεια κατά ηλικία.

Query 6:

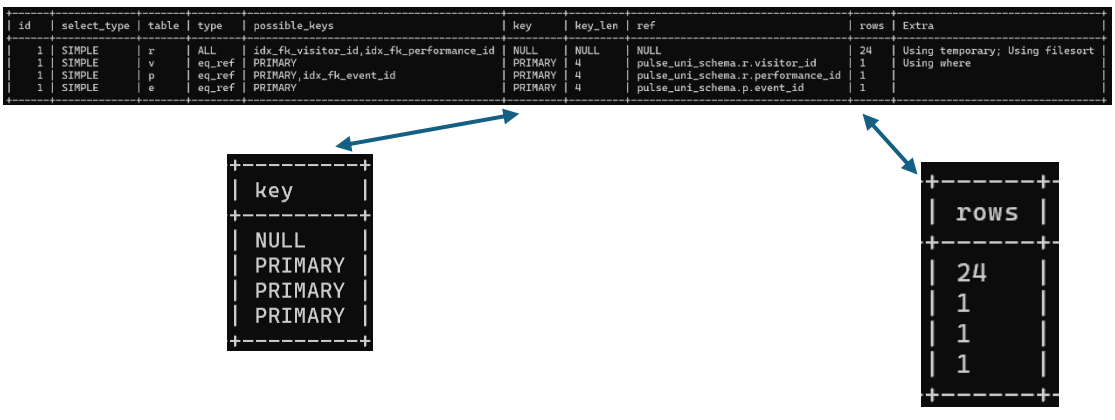
```sql
#-----------------QUERY 6---------------#
DELIMITER //
DROP PROCEDURE IF EXISTS avg_review_of_events_of_a_visitor_1;
CREATE PROCEDURE avg_review_of_events_of_a_visitor_1 (
    IN in_visitor_name VARCHAR(100)
)
BEGIN
    SELECT
    v.visitor_id,
    v.visitor_name,
    e.event_name,
    AVG(r.artist_performance) AS avg_artist_performance_review,
    AVG(r.sound_lighting) AS avg_sound_lighting_review,
    AVG(r.stage_presence) AS avg_stage_presence_review,
    AVG(r.organization) AS avg_overall_impression_review,
    AVG(r.overall_impression) AS avg_overall_impression_review,
    (AVG(r.artist_performance) + AVG(r.sound_lighting)
    + AVG(r.stage_presence) + AVG(r.organization) + AVG(r.overall_impression))/5 AS general_avg
    FROM visitor v IGNORE INDEX (idx_visitor_name)
    JOIN review r ON v.visitor_id = r.visitor_id
    JOIN performance p ON r.performance_id = p.performance_id
    JOIN event e ON p.event_id = e.event_id
    WHERE visitor_name = in_visitor_name
    GROUP BY v.visitor_id, v.visitor_name, e.event_name;
END//
DELIMITER ;
```

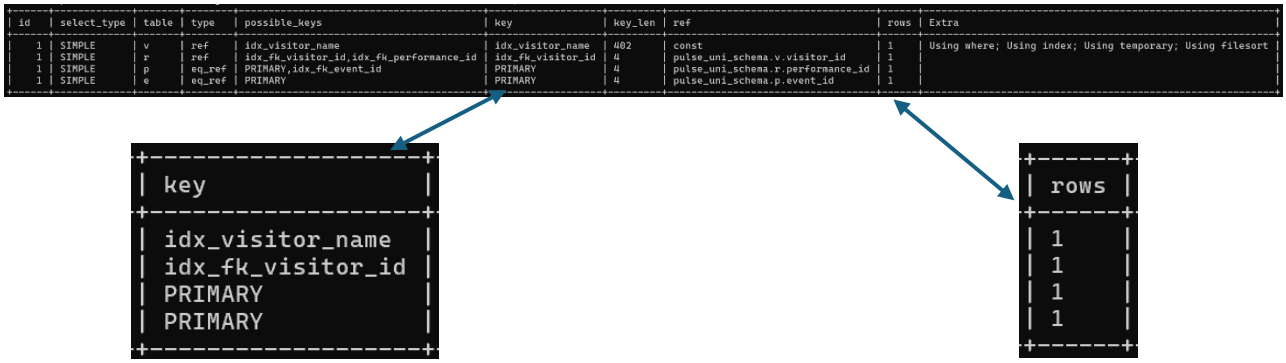
- Το ερώτημα απαντήθηκε με την μορφή διαδικασίας η οποία παίρνει ως όρισμα το όνομα ενός συγκεκριμένου επισκέπτη (visitor_name). Η διαδικασία αυτή ,avg_review_of_events_of_a_visitor_1, επιστρέφει για τον επισκέπτη αυτό (βάσει του ονόματός του) τους μέσους όρους αξιολογήσεων για κάθε event στο οποίο παρευρέθηκε. Υπολογίζονται ξεχωριστοί μέσοι όροι για διάφορα κριτήρια (artist_performance, sound_lighting, stage_presence, organization, overall_impression), καθώς και ένας συνολικός μέσος όρος (general_avg). Η συσχέτιση γίνεται μέσω των πινάκων visitor, review, performance και event, ενώ αγνοείται το index idx_visitor_name.

Query 6/Εναλλακτικό Query Plan:

```
#------------------QUERY 6/ALTERNATIVE QUERY PLAN---------------#
DELIMITER //
DROP PROCEDURE IF EXISTS avg_review_of_events_of_a_visitor_2;
CREATE PROCEDURE avg_review_of_events_of_a_visitor_2 (
    IN in_visitor_name VARCHAR(100)
)
BEGIN
    EXPLAIN SELECT
    v.visitor_id,
    v.visitor_name,
    e.event_name,
    AVG(r.artist_performance) AS avg_artist_performance_review,
    AVG(r.sound_lighting) AS avg_sound_lighting_review,
    AVG(r.stage_presence) AS avg_stage_presence_review,
    AVG(r.organization) AS avg_overall_impression_review,
    AVG(r.overall_impression) AS avg_overall_impression_review,
    (AVG(r.artist_performance) + AVG(r.sound_lighting)
    + AVG(r.stage_presence) + AVG(r.organization) + AVG(r.overall_impression))/5 AS general_avg
    FROM visitor v FORCE INDEX (idx_visitor_name)
    JOIN review r ON v.visitor_id = r.visitor_id
    JOIN performance p ON r.performance_id = p.performance_id
    JOIN event e ON p.event_id = e.event_id
    WHERE visitor_name = in_visitor_name
    GROUP BY v.visitor_id, v.visitor_name, e.event_name;
END//
DELIMITER ;
```

- Επαναλαμβάνεται ο κώδικας του πρώτου πλάνου με επιβολή του βέλτιστου ευρετηρίου idx_visitor_name. Η επιλογή αυτή συνοδεύεται από τα πλεονεκτήματα της χρήσης του ορθού index, όπως αυτά περιγράφηκαν στο query 4, και οδηγούν κατ' επέκταση σε πιο αποδοτική υλοποίηση. Η αποδοτικότητα φανερώνεται εκ νέου με τη βοήθεια των αντίστοιχων traces.





Query 7:

```sql
#------------------QUERY 7----------------#
WITH temp1 AS (
    SELECT staff_id,
        CASE WHEN experience_level = 'Specialist' THEN 1
            WHEN experience_level = 'Beginner' THEN 2
            WHEN experience_level = 'Intermediate' THEN 3
            WHEN experience_level = 'Experienced' THEN 4
            WHEN experience_level = 'Very Experienced' THEN 5
        END AS experience_level_numerated
        FROM staff FORCE INDEX (idx_staff_job)
        WHERE job = 'technical'
),
temp2 AS(
    SELECT
    f.festival_id,
    f.festival_name,
    AVG(CAST(t1.experience_level_numerated AS FLOAT)) AS tech_experience_level_score
    FROM temp1 t1
    JOIN staff_event se ON t1.staff_id = se.staff_id
    JOIN event e ON se.event_id = e.event_id
    JOIN festival f ON e.festival_id = f.festival_id
    GROUP BY f.festival_id, f.festival_name
)
SELECT *
FROM temp2
ORDER BY tech_experience_level_score
LIMIT 1;
```

- Το ερώτημα υπολογίζει ποιο φεστιβάλ είχε τον χαμηλότερο μέσο όρο εμπειρίας τεχνικού προσωπικού. Αρχικά, στο πρώτο CTE (temp1) γίνεται αριθμητική αντιστοίχιση των επιπέδων εμπειρίας για τα τεχνικά μέλη του προσωπικού (χρήση CASE). Στο δεύτερο CTE (temp2) υπολογίζεται ο μέσος όρος (AVG) εμπειρίας για κάθε φεστιβάλ μέσω συσχετίσεων με events και συμμετοχές προσωπικού (staff_event). Η τελική έξοδος επιστρέφει το φεστιβάλ με τον ελάχιστο δείκτη τεχνικής εμπειρίας (tech_experience_level_score), με χρήση ORDER BY, ταξινόμηση κατά αύξουσα σειρά και LIMIT 1, επιλογή της πρώτης εγγραφής.

Query 8:

```
#-------------------QUERY 8----------------#
DELIMITER //
DROP PROCEDURE IF EXISTS free_assistant_staff_for_date;
CREATE PROCEDURE free_assistant_staff_for_date (
    IN specific_date DATE
)
BEGIN
    WITH assistant_staff AS (
        SELECT * FROM staff WHERE job = 'assistant'
    ),

    occupied_assistant_staff AS(
        SELECT s.staff_id
        FROM staff s
        JOIN staff_event se ON s.staff_id = se.staff_id
        JOIN event e FORCE INDEX (idx_event_start_time) ON se.event_id = e.event_id
        WHERE DATE(e.start_time) = specific_date
    )

    SELECT * FROM assistant_staff WHERE staff_id
    NOT IN (
        SELECT staff_id FROM occupied_assistant_staff
    );
END//
DELIMITER ;
```

-Το ερώτημα απαντήθηκε μέσω της υλοποίησης της διαδικασίας free_assistant_staff_for_date, η οποία παίρνει σαν όρισμα την ημερομηνία ενδιαφέροντος και επιστρέφει τα μέλη του προσωπικού με job='*assistant*' που είναι διαθέσιμα σε εκείνη την ημερομηνία. Αρχικά, στο πρώτο CTE (assistant_staff) επιλέγονται όλοι οι βοηθοί, ενώ στο δεύτερο (occupied_assistant_staff) εντοπίζονται όσοι είναι απασχολημένοι εκείνη την ημέρα μέσω συμμετοχών σε events. Τελικά, επιστρέφονται μόνο οι βοηθοί που δεν συμμετέχουν (NOT IN occupied_assistant_staff) σε κάποιο event την επιλεγμένη ημερομηνία, προσδιορίζοντας έτσι το διαθέσιμο προσωπικό.

Query 9 - Query 9/Alternative:

```
#------------------QUERY 9----------------#
WITH visitors_min_appearances AS (
    SELECT
        v.visitor_id,
        v.visitor_name,
        YEAR(e.start_time) AS year,
        COUNT(DISTINCT e.event_id) AS num_events
    FROM visitor v
    JOIN ticket t FORCE INDEX (unique_ticket_per_visitor) ON v.visitor_id = t.visitor_id
    JOIN event e ON t.event_id = e.event_id
    WHERE t.active = FALSE
    GROUP BY v.visitor_id, visitor_name, YEAR(e.start_time)
    HAVING num_events > 3
),
grouped_visitors AS (
    SELECT
        year,
        num_events,
        GROUP_CONCAT(v.visitor_name ORDER BY v.visitor_name) AS visitors
    FROM visitors_min_appearances v
    GROUP BY year, num_events
)
SELECT * FROM grouped_visitors
ORDER BY year, num_events;
```

- Στο πρώτο CTE (visitors_min_appearances), υπολογίζονται οι συμμετοχές ανά επισκέπτη και έτος, και κρατούνται όσοι έχουν παρακολουθήσει (το εισιτήριο έχει κατάσταση active = FALSE) περισσότερα από 3 events (HAVING). Στο δεύτερο CTE (grouped_visitors), ομαδοποιούνται τα αποτελέσματα ανά έτος και πλήθος παρακολουθήσεων, και συγκεντρώνονται τα ονόματα των επισκεπτών με GROUP_CONCAT. Η τελική έξοδος εμφανίζει τους επισκέπτες ανά πλήθος συμμετοχών και έτος, ταξινομημένα ανά έτος και αριθμό events. Υλοποιήθηκε και μια εναλλακτική λύση η οποία παρουσιάζει τα αποτελέσματα χωρίς GROUP_CONCAT

```
#------------------QUERY 9/alternative----------------#
WITH visitors_min_appearances AS (
SELECT
    v.visitor_id,
    v.visitor_name,
    YEAR(e.start_time) AS year,
    COUNT(DISTINCT e.event_id) AS num_events
FROM visitor v
JOIN ticket t FORCE INDEX (unique_ticket_per_visitor) ON v.visitor_id = t.visitor_id
JOIN event e ON t.event_id = e.event_id
WHERE t.active = FALSE
GROUP BY v.visitor_id, YEAR(e.start_time)
HAVING num_events > 3
)
SELECT DISTINCT
vma1.visitor_id,
vma1.visitor_name,
vma1.year,
vma1.num_events
FROM visitors_min_appearances vma1
JOIN visitors_min_appearances vma2 ON vma1.year = vma2.year
AND vma1.num_events = vma2.num_events AND vma1.visitor_id != vma2.visitor_id
ORDER BY year;
```

Query 10 - Query 10/Alternative:

```sql
#------------------QUERY 10----------------#
WITH genre_pairs AS (
    SELECT DISTINCT
        ag1.artist_id,
        LEAST(ag1.genre, ag2.genre) AS genre1,
        GREATEST(ag1.genre, ag2.genre) AS genre2
    FROM artist_genre ag1
    JOIN artist_genre ag2
        ON ag1.artist_id = ag2.artist_id
    WHERE ag1.genre < ag2.genre
),
pair_festivals AS (
    SELECT DISTINCT
        f.festival_id,
        gp.genre1,
        gp.genre2
    FROM genre_pairs gp
    JOIN performance_artist pa ON gp.artist_id = pa.artist_id
    JOIN performance p ON pa.performance_id = p.performance_id
    JOIN event e ON p.event_id = e.event_id
    JOIN festival f ON e.festival_id = f.festival_id
)
SELECT
    genre1,
    genre2,
    COUNT(DISTINCT festival_id) AS festival_count
FROM pair_festivals
GROUP BY genre1, genre2
ORDER BY festival_count DESC
LIMIT 3;
```

- Το ερώτημα εντοπίζει τα τρία πιο συχνά εμφανιζόμενα ζεύγη μουσικών ειδών που συναντώνται σε φεστιβάλ. Αρχικά, στο πρώτο CTE (genre_pairs) δημιουργούνται όλα τα μοναδικά ζεύγη διαφορετικών ειδών που ανήκουν στον ίδιο καλλιτέχνη, εξασφαλίζοντας σταθερή σειρά με LEAST και GREATEST. Στο δεύτερο CTE (pair_festivals) εντοπίζονται τα φεστιβάλ στα οποία συμμετείχαν καλλιτέχνες με τα συγκεκριμένα ζεύγη ειδών. Τέλος, υπολογίζεται πόσες φορές εμφανίστηκε κάθε ζεύγος σε διαφορετικά φεστιβάλ και επιστρέφονται τα 3 πιο συχνά, ταξινομημένα κατά φθίνουσα συχνότητα. Ουσιαστικά επιστρέφονται τα top 3 πιο συχνά εμφανιζόμενα ζευγάρια μουσικών ειδών σε όλα τα φεστιβάλ.

- Υπάρχει, ακόμα, εναλλακτική υλοποίηση που εμφανίζει τα top 3 ζεύγη σε κάθε φεστιβάλ.

```sql
#------------------QUERY 10 PER FESTIVAL SOLUTION 2----------------#
WITH genre_pairs AS (
    SELECT DISTINCT
        ag1.artist_id,
        LEAST(ag1.genre, ag2.genre) AS genre1,
        GREATEST(ag1.genre, ag2.genre) AS genre2
    FROM artist_genre ag1
    JOIN artist_genre ag2
        ON ag1.artist_id = ag2.artist_id
    WHERE ag1.genre < ag2.genre
),
pair_festivals AS (
    SELECT
        f.festival_id,
        f.festival_name,
        gp.genre1,
        gp.genre2,
        COUNT(DISTINCT gp.artist_id) AS artist_count
    FROM genre_pairs gp
    JOIN performance_artist pa ON gp.artist_id = pa.artist_id
    JOIN performance p ON pa.performance_id = p.performance_id
    JOIN event e ON p.event_id = e.event_id
    JOIN festival f ON e.festival_id = f.festival_id
    GROUP BY f.festival_id, f.festival_name, gp.genre1, gp.genre2
),
ranked_pairs AS (
    SELECT *,
        ROW_NUMBER() OVER (
            PARTITION BY festival_id
            ORDER BY artist_count DESC
        ) AS rank
    FROM pair_festivals
)
SELECT
    festival_id,
    festival_name,
    genre1,
    genre2,
    artist_count AS genre_tuple_count
FROM ranked_pairs
WHERE rank <= 3
ORDER BY festival_id, rank;
```

Query 11:

```
#------------------QUERY 11----------------#

WITH tot_apps_of_artists AS (
SELECT
a.artist_id,
a.artist_name,
COUNT(DISTINCT f.festival_id) AS total_artist_appearances
FROM artist a
JOIN performance_artist pa ON a.artist_id = pa.artist_id
JOIN performance p ON pa.performance_id = p.performance_id
JOIN event e ON p.event_id = e.event_id
JOIN festival f ON e.festival_id = f.festival_id
GROUP BY a.artist_id, a.artist_name
)

SELECT *
FROM tot_apps_of_artists
WHERE total_artist_appearances - (SELECT MAX(total_artist_appearances) FROM tot_apps_of_artists ) <= -5
ORDER BY total_artist_appearances DESC;
```

-Το ερώτημα εντοπίζει καλλιτέχνες που έχουν εμφανιστεί σε τουλάχιστον 5 φεστιβάλ λιγότερα από τον πιο ενεργό καλλιτέχνη. Στο CTE tot_apps_of_artists υπολογίζεται ο συνολικός αριθμός εμφανίσεων κάθε καλλιτέχνη σε φεστιβάλ. Στη συνέχεια, επιλέγονται μόνο εκείνοι των οποίων οι εμφανίσεις υπολείπονται κατά 5 ή περισσότερα από τη μέγιστη τιμή. Η τελική έξοδος ταξινομείται κατά φθίνουσα σειρά εμφανίσεων.

Query 12 - Query 12/Alternative:

```
#------------------QUERY 12----------------#
DELIMITER //
DROP PROCEDURE IF EXISTS staff_info;
CREATE PROCEDURE staff_info (
    IN in_festival_id INT
)
BEGIN
SELECT
f.festival_id,
f.festival_name,
s.*,
DATE(e.start_time) AS festival_day
FROM staff s
JOIN staff_event se ON s.staff_id = se.staff_id
JOIN event e ON se.event_id = e.event_id
JOIN festival f ON e.festival_id = f.festival_id
WHERE f.festival_id = in_festival_id
ORDER BY festival_day, s.job, s.staff_id;
END//
DELIMITER ;
```

- Δημιουργήθηκε διαδικασία staff_info, η οποία παίρνει σαν παράμετρο το id του φετσιβάλ από το οποίο θέλουμε αναλυτική πληροφορία για το προσωπικό του. Συγκεκριμένα, γίνεται συσχέτιση των πινάκων staff, staff_event, event και festival, ώστε να εμφανιστούν τα στοιχεία του προσωπικού μαζί με την ημερομηνία διεξαγωγής των αντίστοιχων παραστάσεων τις οποίες στελεχώνουν. Το αποτέλεσμα ταξινομείται πρώτα κατά ημερομηνία (festival_day), έπειτα κατά ρόλο (job) και τέλος κατά staff_id. Η πληροφορία για το προσωπικό παρουσιάζεται σε επίπεδο ατόμου.

```
#------------------QUERY 12/alternative----------------#
DELIMITER //
DROP PROCEDURE IF EXISTS staff_info;
CREATE PROCEDURE staff_info (
    IN in_festival_id INT
)
BEGIN
    SELECT
        f.festival_id,
        f.festival_name,
        DATE(e.start_time) AS festival_day,
        SUM(CASE WHEN s.job = 'technical' THEN 1 ELSE 0 END) AS techincal_staff,
        SUM(CASE WHEN s.job = 'security' THEN 1 ELSE 0 END) AS security_staff,
        SUM(CASE WHEN s.job = 'assistant' THEN 1 ELSE 0 END) AS assistant_staff
    FROM staff s
    JOIN staff_event se ON s.staff_id = se.staff_id
    JOIN event e ON se.event_id = e.event_id
    JOIN festival f ON e.festival_id = f.festival_id
    WHERE f.festival_id = in_festival_id
    GROUP BY f.festival_id, f.festival_name, festival_day;
END //
DELIMITER ;
```

- Η εναλλακτική απάντηση μας δίνει το πλήθος του προσωπικού που απαιτείται για κάθε μέρα του φεστιβάλ διαχωρισμένο ανάλογα με το job. Στην υλοποίηση αυτή δεν παρουσιάζεται αναλυτική πληροφορία για κάθε μέλος του προσωπικού ξεχωριστά. Ο διαχωρισμός του προσωπικού γίνεται με την βοήθεια της CASE. Υπάρχει και τρίτη υλοποίηση που παρουσιάζει την ίδια πληροφορία με την τελευταία, αλλά για όλα τα φεστιβάλ και όχι μόνο για ένα.

Query 13:

```
#------------------QUERY 13---------------#
SELECT
a.artist_id,
a.artist_name,
COUNT(DISTINCT l.continent) AS num_continents
FROM artist a
JOIN performance_artist pa ON a.artist_id = pa.artist_id
JOIN performance p ON pa.performance_id = p.performance_id
JOIN event e ON p.event_id = e.event_id
JOIN festival f ON e.festival_id = f.festival_id
JOIN location l ON f.location_id = l.location_id
GROUP BY a.artist_id, a.artist_name
HAVING num_continents>=3
ORDER BY num_continents DESC;
```

- Γίνεται χρήση πολλών JOIN για τον προσδιορισμό της γεωγραφικής θέσης των παραστάσεων κάθε καλλιτέχνη. Τα αποτελέσματα ομαδοποιούνται (GROUP BY) ανά καλλιτέχνη και μετρούνται (COUNT) οι διαφορετικοί (DISTINCT) ήπειροι (continent) στις οποίες έχει εμφανιστεί κάθε καλλιτέχνης.

Query 14:

```
#------------------QUERY 14---------------#
WITH gen_year_num AS(
SELECT
ag.genre,
YEAR(p.start_time) AS year,
COUNT(DISTINCT p.performance_id) AS num_genre_annually
FROM artist_genre ag
JOIN performance_artist pa ON ag.artist_id = pa.artist_id
JOIN performance p ON pa.performance_id = p.performance_id
GROUP BY genre,YEAR(p.start_time)
HAVING COUNT(DISTINCT p.performance_id) >= 3
)
SELECT
gyn1.genre,
gyn1.year AS year1,
gyn2.year AS year2,
gyn1.num_genre_annually
FROM gen_year_num gyn1
JOIN gen_year_num gyn2
    ON gyn1.genre = gyn2.genre AND gyn1.year = gyn2.year + 1
WHERE gyn1.num_genre_annually = gyn2.num_genre_annually;
```

- Το ερώτημα βασίζεται στη χρήση ενός CTE (Common Table Expression) με το όνομα gen_year_num. Στο πρώτο μέρος πραγματοποιείται σύνδεση των πινάκων artist_genre, performance_artist και performance και μετριέται ο αριθμός των διαφορετικών εμφανίσεων (COUNT(DISTINCT p.performance_id)) για κάθε genre και έτος (YEAR(p.start_time) με χρήση GROUP BY. Ύστερα εφαρμόζεται φίλτρο HAVING >= 3, ώστε να διατηρηθούν μόνο τα ζεύγη είδους-έτους με τουλάχιστον 3 εμφανίσεις. Στο κύριο ερώτημα πραγματοποιείται self-join του gen_year_num, ώστε να συγκριθούν τα δεδομένα διαδοχικών ετών για το ίδιο είδος εφαρμόζοντας παράλληλα τον έλεγχο για το αν ο αριθμός εμφανίσεων είναι ίσος μεταξύ δύο συνεχόμενων ετών (gyn1.year = gyn2.year + 1 και gyn1.num_genre_annually = gyn2.num_genre_annually). Τέλος προβάλλονται τα είδη, τα δύο συνεχόμενα έτη και ο αριθμός εμφανίσεων

Query 15 - Query 15/Alternative:

```
#------------------QUERY 15----------------#
SELECT
v.visitor_id,
v.visitor_name,
a.artist_id,
a.artist_name,
(r.artist_performance + r.sound_lighting + r.stage_presence
+ r.organization + r.overall_impression) AS overall_score
FROM visitor v
JOIN review r ON v.visitor_id = r.visitor_id
JOIN performance_artist pa ON r.performance_id = pa.performance_id
JOIN artist a ON pa.artist_id = a.artist_id
ORDER BY overall_score DESC
LIMIT 5;
```

- Η υλοποίηση αυτή παρουσιάζει τις top 5 reviews από όλα τα χρόνια του φεστιβάλ και συνοδεύεται από πληροφορία για visitor που έκανε το review και για τον καλλιτέχνη που αξιολόγησε. Η συνολική βαθμολογία προκύπτει αθροίζοντας όλα τα πεδία του review και επιπλέον αν ένας επισκέπτης έχει κάνει πολλαπλή αξιολόγηση για κάποιον καλλιτέχνη τότε εμφανίζεται ο μέσος όρος των επιμέρους αξιολογήσεων.

- Στην προσπάθεια μας να αποσαφηνίσουμε το πραγματικό ζητούμενο της άσκησης προέκυψαν και διαφορετικές προσεγγίσεις του ερωτήματος. Παραθέτουμε μία από αυτές η οποία βρίσκει τις top 5 reviews ανά καλλιτέχνη, και σε περίπτωση πολλαπλής αξιολόγησης από τον ίδιο επισκέπτη προσμετράται ο μέσος όρος αυτών.

```sql
#------------------QUERY 15 / first alternative----------------#
WITH visitor_artist_avg AS (
    SELECT
        a.artist_id,
        a.artist_name,
        v.visitor_id,
        v.visitor_name,
        AVG(r.artist_performance) AS avg_artist_score
    FROM review r
    JOIN visitor v ON r.visitor_id = v.visitor_id
    JOIN performance_artist pa ON r.performance_id = pa.performance_id
    JOIN artist a ON pa.artist_id = a.artist_id
    GROUP BY a.artist_id, a.artist_name, v.visitor_id, v.visitor_name
),
ranked_reviews AS (
    SELECT *,
        ROW_NUMBER() OVER (
            PARTITION BY artist_id
            ORDER BY avg_artist_score DESC
        ) AS rn
    FROM visitor_artist_avg
)
SELECT
artist_id,
artist_name,
visitor_id,
visitor_name,
avg_artist_score
FROM ranked_reviews
WHERE rn <= 5
ORDER BY artist_id, avg_artist_score DESC;
```

- Τέλος άλλες εξίσου ενδιαφέρουσες απαντήσεις για το ερώτημα 15 βρίσκονται στα αντίστοιχα αρχεία κώδικα.

Χρήσιμες Ιστοσελίδες

draw.io: για τη σχεδίαση του E/R Διαγράμματος.
dbdiagram.io: για τη δημιουργία του Relational Διαγράμματος.
Stackedit.io: για τη συγγραφή του README.md